DATA AGGREGATION SERVER FOR MANAGING A MULTI-DIMENSIONAL DATABASE AND DATABASE MANAGEMENT SYSTEM HAVING DATA AGGREGATION SERVER INTEGRATED THEREIN

Inventors:
Reuven Bakalash
Guy Shaked
Joseph Caspi

RELATED CASES

This is a Continuation-in-part of: copending U.S. Application Serial No. 09/514,611 entitled "Stand-Alone Cartridge-Style Data Aggregation Server And Method of And System For Managing Multi-Dimensional Databases using the Same", filed February 28, 2000, and U.S. Application Serial No. 09/634,748 entitled "Relational Database Management System Having Integrated Non-Relational Multi-Dimensional Data Store of Aggregated Data Elements" filed August 9, 2000; each said Application being commonly owned by HyperRoll, Limited, and incorporated herein by reference in its entirety.

BACKGROUND OF THE INVENTION

Field of Invention

The present invention relates to a method of and system for aggregating data elements in a multi-dimensional database (MDDB) supported upon a computing platform and also to provide an improved method of and system for managing data elements within a MDDB during on-line analytical processing (OLAP) operations and as an integral part of a database management system.

Brief Description Of The State Of The Art

The ability to act quickly and decisively in today's increasingly competitive marketplace is critical to the success of organizations. The volume of information that is available to corporations is rapidly increasing and frequently overwhelming. Those organizations that will effectively and efficiently manage these tremendous volumes of data, and use the information to make business decisions, will realize a significant competitive advantage in the marketplace.

Data warehousing, the creation of an enterprise-wide data store, is the first step towards managing these volumes of data. The Data Warehouse is becoming an integral part of many information delivery systems because it provides a single, central location where a

reconciled version of data extracted from a wide variety of operational systems is stored. Over the last few years, improvements in price, performance, scalability, and robustness of open computing systems have made data warehousing a central component of Information Technology CIT strategies. Details on methods of data integration and constructing data warehouses can be found in the white paper entitled "Data Integration: The Warehouse Foundation" by Louis Rollleigh and Joe Thomas, published at http://www.acxiom.com/whitepapers/wp-11.asp .

Building a Data Warehouse has its own special challenges (e.g. using common data model, common business dictionary, etc.) and is a complex endeavor. However, just having a Data Warehouse does not provide organizations with the often-heralded business benefits of data warehousing. To complete the supply chain from transactional systems to decision maker, organizations need to deliver systems that allow knowledge workers to make strategic and tactical decisions based on the information stored in these warehouses. These decision support systems are referred to as On-Line Analytical Processing (OLAP) systems. OLAP systems allow knowledge workers to intuitively, quickly, and flexibly manipulate operational data using familiar business terms, in order to provide analytical insight into a particular problem or line of inquiry. For example, by using an OLAP system, decision makers can "slice and dice" information along a customer (or business) dimension, and view business metrics by product and through time. Reports can be defined from multiple perspectives that provide a high-level or detailed view of the performance of any aspect of the business. Decision makers can navigate throughout their database by drilling down on a report to view elements at finer levels of detail, or by pivoting to view reports from different perspectives. To enable such full-functioned business analyses, OLAP systems need to (1) support sophisticated analyses, (2) scale to large numbers of dimensions, and (3) support analyses against large atomic data sets. These three key requirements are discussed further below. Decision makers use key performance metrics to evaluate the operations within their domain, and OLAP systems need to be capable of delivering these metrics in a user-customizable format. These metrics may be obtained from the transactional databases pre-calculated and stored in the database, or generated on demand during the query process. Commonly used metrics include:

(1) Multidimensional Ratios (e.g. Percent to Total) - "Show me the contribution to weekly sales and category profit made by all items sold in the Northwest stores between July 1 and July 14."

(2) Comparisons (e.g. Actual vs. Plan, This Period vs. Last Period) - "Show me the sales to plan percentage variation for this year and compare it to that of the previous year to identify planning discrepancies."

(3) Ranking and Statistical Profiles (e.g. Top N/Bottom N, 70/30, Quartiles) - "Show me sales, profit and average call volume per day for my 20 most profitable salespeople, who are in the top 30% of the worldwide sales."

(4) Custom Consolidations - "Show me an abbreviated income statement by quarter for the last two quarters for my Western Region operations."

Knowledge workers analyze data from a number of different business perspectives or dimensions. As used hereinafter, a dimension is any element or hierarchical combination of elements in a data model that can be displayed orthogonally with respect to other combinations of elements in the data model. For example, if a report lists sales by week, promotion, store, and department, then the report would be a slice of data taken from a four-dimensional data model.

Target marketing and market segmentation applications involve extracting highly qualified result sets from large volumes of data. For example, a direct marketing organization might want to generate a targeted mailing list based on dozens of characteristics, including purchase frequency, size of the last purchase, past buying trends, customer location, age of customer, and gender of customer. These applications rapidly increase the dimensionality requirements for analysis.

The number of dimensions in OLAP systems range from a few orthogonal dimensions to hundreds of orthogonal dimensions. Orthogonal dimensions in an exemplary OLAP application might include Geography, Time, and Products.

Atomic data refers to the lowest level of data granularity required for effective decision making. In the case of a retail merchandising manager, "atomic data" may refer to information by store, by day, and by item. For a banker, atomic data may be information by account, by transaction, and by branch. Most organizations implementing OLAP systems find themselves needing systems that can scale to tens, hundreds, and even thousands of gigabytes of atomic information.

As OLAP systems become more pervasive and are used by the majority of the enterprise, more data over longer time frames will be included in the data store (i.e. data warehouse), and the size of the database will increase by at least an order of magnitude. Thus, OLAP systems need to be able to scale from present to near-future volumes of data.

In general, OLAP systems need to (1) support the complex analysis requirements of decision-makers, (2) analyze the data from a number of different perspectives (i.e. business dimensions), and (3) support complex analyses against large input (atomic-level) data sets from a Data Warehouse maintained by the organization using a relational database management system (RDBMS).

Vendors of OLAP systems classify OLAP Systems as either Relational OLAP (ROLAP) or Multidimensional OLAP (MOLAP) based on the underlying architecture thereof. Thus, there are two basic architectures for On-Line Analytical Processing systems: the ROLAP Architecture, and the MOLAP architecture.

The Relational OLAP (ROLAP) system accesses data stored in a Data Warehouse to provide OLAP analyses. The premise of ROLAP is that OLAP capabilities are best provided directly against the relational database, i.e. the Data Warehouse.

The ROLAP architecture was invented to enable direct access of data from Data Warehouses, and therefore support optimization techniques to meet batch window requirements and provide fast response times. Typically, these optimization techniques include application-level table partitioning, pre-aggregate inferencing, denormalization support, and the joining of multiple fact tables.

A typical prior art ROLAP system has a three-tier or layer client/server architecture. The "database layer" utilizes relational databases for data storage, access, and retrieval processes. The "application logic layer" is the ROLAP engine which executes the multidimensional reports from multiple users. The ROLAP engine integrates with a variety of "presentation layers," through which users perform OLAP analyses.

After the data model for the data warehouse is defined, data from on-line transaction-processing (OLTP) systems is loaded into the relational database management system (RDBMS). If required by the data model, database routines are run to pre-aggregate the data within the RDBMS. Indices are then created to optimize query access times. End users submit multidimensional analyses to the ROLAP engine, which then dynamically transforms the requests into SQL execution plans. The SQL execution plans are submitted to the relational database for processing, the relational query results are cross-tabulated, and a multidimensional result data set is returned to the end user. ROLAP is a fully dynamic architecture capable of utilizing pre-calculated results when they are available, or dynamically generating results from atomic information when necessary.

Multidimensional OLAP (MOLAP) systems utilize a proprietary multidimensional database (MDDB) to provide OLAP analyses. The MDDB is logically organized as a multidimensional array (typically referred to as a multidimensional cube or hypercube or cube) whose rows/columns each represent a different dimension (i.e., relation). A data value is associated with each combination of dimensions (typically referred to as a "coordinate"). The main premise of this architecture is that data must be stored multidimensionally to be accessed and viewed multidimensionally.

As shown in Fig. 1B, prior art MOLAP systems have an Aggregation, Access and Retrieval module which is responsible for all data storage, access, and retrieval processes, including data aggregration (i.e. preaggregation) in the MDDB. As shown in Fig. 1B, the base data loader is fed with base data, in the most detailed level, from the Data Warehouse, into the Multi-Dimensional Data Base (MDDB). On top of the base data, layers of aggregated data are built-up by the Aggregation program, which is part of the Aggregation, Access and Retrieval module. As indicated in this figure, the application logic module is responsible for the execution of all OLAP requests/queries (e.g. ratios, ranks, forecasts, exception scanning, and slicing and dicing) of data within the MDDB. The presentation module integrates with the application logic module and provides an interface, through which the end users view and request OLAP analyses on their client machines which may be web-

enabled through the infrastructure of the Internet. The client/server architecture of a MOLAP system allows multiple users to access the same multidimensional database (MDDB).

Information (i.e. basic data) from a variety of operational systems within an enterprise, comprising the Data Warehouse, is loaded into a prior art multidimensional database (MDDB) through a series of batch routines. The Express™ server by the Oracle Corporation is exemplary of a popular server which can be used to carry out the data loading process in prior art MOLAP systems. As shown in Fig. 2B, an exemplary 3-D MDDB is schematically depicted, showing geography, time and products as the "dimensions" of the database. The multidimensional data of the MDDB is logically organized in an array structure, as shown in Fig. 2C. Physically, the Express™ server stores data in pages (or records) of an information file. Pages contain 512, or 2048, or 4096 bytes of data, depending on the platform and release of the Express™ server. In order to look up the physical record address from the database file recorded on a disk or other mass storage device, the Express™ server generates a data structure referred to as a "Page Allocation Table (PAT)". As shown in Fig. 2D, the PAT tells the Express™ server the physical record number that contains the page of data. Typically, the PAT is organized in pages. The simplest way to access a data element in the MDDB is by calculating the "offset" using the additions and multiplications expressed by a simple formula:

$$Offset = Months + Product * (\#\ of\_Months) + City * (\#\ of\_Months * \#\ of\_Products)$$

During an OLAP session, the response time of a multidimensional query on a prior art MDDB depends on how many cells in the MDDB have to be added "on the fly". As the number of dimensions in the MDDB increases linearly, the number of the cells in the MDDB increases exponentially. However, it is known that the majority of multidimensional queries deal with summarized high level data. Thus, as shown in Figs. 3A and 3B, once the atomic data (i.e. "basic data") has been loaded into the MDDB, the general approach is to perform a series of calculations in batch in order to aggregate (i.e. pre-aggregate) the data elements along the orthogonal dimensions of the MDDB and fill the array structures thereof. For example, revenue figures for all retail stores in a particular state (i.e. New York) would be added together to fill the state level cells in the MDDB. After the array structure in the database has been filled, integer-based indices are created and hashing algorithms are used to improve query access times. Pre-aggregation of dimension D0 is always performed along the cross-section of the MDDB along the D0 dimension.

As shown in Figs. 3C1 and 3C2, the raw data loaded into the MDDB is primarily organized at its lowest dimensional hierarchy, and the results of the pre-aggregations are stored in the neighboring parts of the MDDB.

As shown in Fig. 3C2, along the TIME dimension, *weeks* are the aggregation results of *days*, *months* are the aggregation results of *weeks*, and *quarters* are the aggregation results of *months*. While not shown in the figures, along the GEOGRAPHY dimension, states are the

aggregation results of cities, countries are the aggregation results of states, and continents are the aggregation results of countries. By *pre-aggregating* (i.e. consolidating or compiling) all logical subtotals and totals along all dimensions of the MDDB, it is possible to carry out real-time MOLAP operations using a multidimensional database (MDDB) containing both basic (i.e. atomic) and pre-aggregated data. Once this compilation process has been completed, the MDDB is ready for use. Users request OLAP reports by submitting queries through the OLAP Application interface (e.g. using web-enabled client machines), and the application logic layer responds to the submitted queries by retrieving the stored data from the MDDB for display on the client machine.

Typically, in MDDB systems, the aggregated data is very sparse, tending to explode as the number of dimension grows and dramatically slowing down the retrieval process (as described in the report entitled "Database Explosion: The OLAP Report", http://www.olapreport.com/DatabaseExplosion.htm , incorporated herein by reference). Quick and on line retrieval of queried data is critical in delivering on-line response for OLAP queries. Therefore, the data structure of the MDDB, and methods of its storing, indexing and handling are dictated mainly by the need of fast retrieval of massive and sparse data.

Different solutions for this problem are disclosed in the following US Patents, each of which is incorporated herein by reference in its entirety:

- 5,822,751 "Efficient Multidimensional Data Aggregation Operator Implementation"
- 5,805,885 "Method And System For Aggregation Objects"
- 5,781,896 "Method And System For Efficiently Performing Database Table Aggregation Using An Aggregation Index"
- 5,745,764 "Method And System For Aggregation Objects"

In all the prior art OLAP servers, the process of storing, indexing and handling MDDB utilize complex data structures to largely improve the retrieval speed, as part of the querying process, at the cost of slowing down the storing and aggregation. The query-bounded structure, that must support fast retrieval of queries in a restricting environment of high sparcity and multi-hierarchies, is not the optimal one for fast aggregation.

In addition to the aggregation process, the Aggregation, Access and Retrieval module is responsible for all data storage, retrieval and access processes. The Logic module is responsible for the execution of OLAP queries. The Presentation module intermediates between the user and the logic module and provides an interface through which the end users view and request OLAP analyses. The client/server architecture allows multiple users to simultaneously access the multidimensional database.

In summary, general system requirements of OLAP systems include: (1) supporting sophisticated analysis, (2) scaling to large number of dimensions, and (3) supporting analysis against large atomic data sets.

MOLAP system architecture is capable of providing analytically sophisticated reports and analysis functionality. However, requirements (2) and (3) fundamentally limit MOLAP's capability, because to be effective and to meet end-user requirements, MOLAP databases need a high degree of aggregation.

By contrast, the ROLAP system architecture allows the construction of systems requiring a low degree of aggregation, but such systems are significantly slower than systems based on MOLAP system architecure principles. The resulting long aggregation times of ROLAP systems impose severe limitations on its volumes and dimensional capabilities.

The graphs plotted in Fig. 5 clearly indicate the computational demands that are created when searching an MDDB during an OLAP session, where answers to queries are presented to the MOLAP system, and answers thereto are solicited often under real-time constraints. However, prior art MOLAP systems have limited capabilities to dynamically create data aggregations or to calculate business metrics that have not been precalculated and stored in the MDDB.

The large volumes of data and the high dimensionality of certain market segmentation applications are orders of magnitude beyond the limits of current multidimensional databases.

ROLAP is capable of higher data volumes. However, the ROLAP architecture, despite its high volume and dimensionality superiority, suffers from several significant drawbacks as compared to MOLAP:

- Full aggregation of large data volumes are very time consuming, otherwise, partial aggregation severely degrades the query response.
- It has a slower query response
- It requires developers and end users to know SQL
- SQL is less capable of the sophisticated analytical functionality necessary for OLAP
- ROLAP provides limited application functionality

Thus, improved techniques for data aggregation within MOLAP systems would appear to allow the number of dimensions of and the size of atomic (i.e. basic) data sets in the MDDB to be significantly increased, and thus increase the usage of the MOLAP system architecture.

Also, improved techniques for data aggregation within ROLAP systems would appear to allow for maximized query performance on large data volumes, and reduce the time of partial aggregations that degrades query response, and thus generally benefit ROLAP system architectures.

Thus, there is a great need in the art for an improved way of and means for aggregating data elements within a multi-dimensional database (MDDB), while avoiding the shortcomings and drawbacks of prior art systems and methodologies.

Modern operational and informational database systems, as described above, typically use a database management system (DBMS) (such as an RDBMS system, object database system, or object/relational database system) as a repository for storing data and querying the data. FIG. 14 illustrates a data warehouse-OLAP domain that utilizes the prior art approaches described above. The data warehouse is an enterprise-wide data store. It is becoming an integral part of many information delivery systems because it provides a single, central location wherein a reconciled version of data extracted from a wide variety of operational systems is stored. Details on methods of data integration and constructing data warehouses can be found in the white paper entitled "Data Integration: The Warehouse Foundation" by Louis Rollleigh and Joe Thomas, published at http://www.acxiom.com/whitepapers/wp-11.asp.

Building a Data Warehouse has its own special challenges (e.g. using common data model, common business dictionary, etc.) and is a complex endeavor. However, just having a Data Warehouse does not provide organizations with the often-heralded business benefits of data warehousing. To complete the supply chain from transactional systems to decision maker, organizations need to deliver systems that allow knowledge workers to make strategic and tactical decisions based on the information stored in these warehouses. These decision support systems are referred to as On-Line Analytical Processing (OLAP) systems. Such OLAP systems are commonly classified as Relational OLAP systems or Multi-Dimensional OLAP systems as described above.

The Relational OLAP (ROLAP) system accesses data stored in a relational database (which is part of the Data Warehouse) to provide OLAP analyses. The premise of ROLAP is that OLAP capabilities are best provided directly against the relational database. The ROLAP architecture was invented to enable direct access of data from Data Warehouses, and therefore support optimization techniques to meet batch window requirements and provide fast response times. Typically, these optimization techniques include application-level table partitioning, pre-aggregate inferencing, denormalization support, and the joining of multiple fact tables.

As described above, a typical ROLAP system has a three-tier or layer client/server architecture. The "database layer" utilizes relational databases for data storage, access, and retrieval processes. The "application logic layer" is the ROLAP engine which executes the multidimensional reports from multiple users. The ROLAP engine integrates with a variety of "presentation layers," through which users perform OLAP analyses. After the data model for the data warehouse is defined, data from on-line transaction-processing (OLTP) systems is loaded into the relational database management system (RDBMS). If required by the data model, database routines are run to pre-aggregate the data within the RDBMS. Indices are then created to optimize query access times. End users submit multidimensional analyses to the ROLAP engine, which then dynamically transforms the requests into SQL execution plans. The SQL execution plans are submitted to the relational database for processing, the relational query results are cross-tabulated, and a multidimensional result data set is returned to the end user. ROLAP is a fully dynamic architecture capable of utilizing pre-calculated

results when they are available, or dynamically generating results from the raw information when necessary.

The Multidimensional OLAP (MOLAP) systems utilize a proprietary multidimensional database (MDDB) (or "cube") to provide OLAP analyses. The main premise of this architecture is that data must be stored multidimensionally to be accessed and viewed multidimensionally. Such MOLAP systems provide an interface that enables users to query the MDDB data structure such that users can "slice and dice" the aggregated data. As shown in Fig. 15, such MOLAP systems have an aggregation engine which is responsible for all data storage, access, and retrieval processes, including data aggregation (i.e. pre-aggregation) in the MDDB, and an analytical processing and GUI module responsible for interfacing with a user to provide analytical analysis, query input, and reporting of query results to the user. In a relational database, data is stored in tables. In contrast, the MDDB is a non-relational data structure - it uses other data structures, either instead of or in addition to tables - to store data.

There are other application domains where there is a great need for improved methods of and apparatus for carrying out data aggregation operations. For example, modern operational and informational databases represent such domains. As described above, modern operational and informational databases typically utilize a relational database system (RDBMS) as a repository for storing data and querying data. FIG. 16A illustrates an exemplary table in an RDBMS; and FIGS. 16B and 16C illustrate operators (queries) on the table of FIG. 16A, and the result of such queries, respectively. The operators illustrated in FIGS. 16B and 16C are expressed as Structured Query Language (SQL) statements as is conventional in the art.

The choice of using a RDBMS as the data repository in information database systems naturally stems from the realities of SQL standardization, the wealth of RDBMS-related tools, and readily available expertise in RDBMS systems. However, the querying component of RDBMS technology suffers from performance and optimization problems stemming from the very nature of the relational data model. More specifically, during query processing, the relational data model requires a mechanism that locates the raw data elements that match the query. Moreover, to support queries that involve aggregation operations, such aggregation operations must be performed over the raw data elements that match the query. For large multi-dimensional databases, a naive implementation of these operations involves computational intensive table scans that leads to unacceptable query response times.

In order to better understand how the prior art has approached this problem, it will be helpful to briefly describe the relational database model. According to the relational database model, a relational database is represented by a logical schema and tables that implement the schema. The logical schema is represented by a set of templates that define one or more dimensions (entities) and attributes associated with a given dimension. The attributes associated with a given dimension includes one or more attributes that distinguish it from every other dimension in the database (a dimension identifier). Relationships amongst dimensions are formed by joining attributes. The data structure that represents the

set of templates and relations of the logical schema is typically referred to as a catalog or dictionary. Note that the logical schema represents the relational organization of the database, but does not hold any fact data per se. This fact data is stored in tables that implement the logical schema.

Star schemas are frequently used to represent the logical structure of a relational database. The basic premise of star schemas is that information can be classified into two groups: facts and dimensions. Facts are the core data elements being analyzed. For example, units of individual item sold are facts, while dimensions are attributes about the facts. For example, dimensions are the product types purchased and the data purchase. Business questions against this schema are asked looking up specific facts (UNITS) through a set of dimensions (MARKETS, PRODUCTS, PERIOD). The central fact table is typically much larger than any of its dimension tables.

An exemplary star schema is illustrated in FIG. 17A for suppliers (the "Supplier" dimension) and parts (the "Parts" dimension) over time periods (the "Time-Period" dimension). It includes a central fact table "Supplied-Parts" that relates to multiple dimensions - the "Supplier", "Parts" and "Time-Period" dimensions. FIG. 17B illustrates the tables used to implement the star schema of FIG. 17A. More specifically, these tables include a central fact table and a dimension table for each dimension in the logical schema of FIG. 17A. A given dimension table stores rows (instances) of the dimension defined in the logical schema. For the sake of description, FIG. 17B illustrates the dimension table for the "Time-Period" dimension only. Similar dimension tables for the "Supplier" and "Part" dimensions (not shown) are also included in such an implementation. Each row within the central fact table includes a multi-part key associated with a set of facts (in this example, a number representing a quantity). The multi-part key of a given row (values stored in the S#,P#,TP# fields as shown) points to rows (instances) stored in the dimension tables described above. A more detailed description of star schemas and the tables used to implement star schemas may be found in C.J. Date, "An Introduction to Database Systems," Seventh Edition, Addison-Wesley, 2000, pp. 711-715, herein incorporated by reference in its entirety.

When processing a query, the tables that implement the schema are accessed to retrieve the facts that match the query. For example, in a star schema implementation as described above, the facts are retrieved from the central fact table and/or the dimension tables. Locating the facts that match a given query involves one or more join operations. Moreover, to support queries that involve aggregation operations, such aggregation operations must be performed over the facts that match the query. For large multi-dimensional databases, a naive implementation of these operations involves computational intensive table scans that typically leads to unacceptable query response times. Moreover, since the fact tables are pre-summarized and aggregated along business dimensions, these tables tend to be very large. This point becomes an important consideration of the performance issues associated with star schemas. A more detailed discussion of the performance issues (and proposed approaches that address such issues) related to joining and aggregation of star schema is now set forth.

The first performance issue arises from computationally intensive table scans that are performed by a naive implementation of data joining. Indexing schemes may be used to bypass these scans when performing joining operations. Such schemes include B-tree indexing, inverted list indexing and aggregate indexing. A more detailed description of such indexing schemes can be found in "The Art of Indexing", Dynamic Information Systems Corporation, October 1999, available at http://www.disc.com/artindex.pdf. All of these indexing schemes replaces table scan operations (involved in locating the data elements that match a query) with one ore more index lookup operation. Inverted list indexing associates an index with a group of data elements, and stores (at a location identified by the index) a group of pointers to the associated data elements. During query processing, in the event that the query matches the index, the pointers stored in the index are used to retrieve the corresponding data elements pointed therefrom. Aggregation indexing integrates an aggregation index with an inverted list index to provide pointers to raw data elements that require aggregation, thereby providing for dynamic summarization of the raw data elements that match the user-submitted query.

These indexing schemes are intended to improve join operations by replacing table scan operations with one or more index lookup operation in order to locate the data elements that match a query. However, these indexing schemes suffer from various performance issues as follows:

- Since the tables in the star schema design typically contain the entire hierarchy of attributes (e.g. in a PERIOD dimension, this hierarchy could be day>week>month>quarter>year), a multipart key of day, week, month, quarter, year has to be created; thus, multiple meta-data definitions are required (one of each key component) to define a single relationship; this adds to the design complexity, and sluggishness in performance.
- Addition or deletion of levels in the hierarchy will require physical modification of the fact table, which is time consuming process that limits flexibility.
- Carrying all the segments of the compound dimensional key in the fact table increases the size of the index, thus impacting both performance and scalability.

Another performance issue arises from dimension tables that contain multiple hierarchies. In such cases, the dimensional table often includes a level of hierarchy indicator for every record. Every retrieval from fact table that stores details and aggregates must use the indicator to obtain the correct result, which impacts performance. The best alternative to using the level indicator is the snowflake schema. In this schema aggregate tables are created separately from the detail tables. In addition to the main fact tables, snowflake schema contains separate fact tables for each level of aggregation. Notably, the snowflake schema is even more complicated than a star schema, and often requires multiple SQL statements to get the results that are required.

Another performance issue arises from the pairwise join problem. Traditional RDBMS engines are not design for the rich set of complex queries that are issued against a star schema. The need to retrieve related information from several tables in a single query –

"join processing" – is severely limited. Many RDBMSs can join only two tables at a time. If a complex join involves more than two tables, the RDBMS needs to break the query into a series of pairwise joins. Selecting the order of these joins has a dramatic performance impact. There are optimizers that spend a lot of CPU cycles to find the best order in which to execute those joins. Unfortunately, because the number of combinations to be evaluated grows exponentially with the number of tables being joined, the problem of selecting the best order of pairwise joins rarely can be solved in a reasonable amount of time.

Moreover, because the number of combinations is often too large, optimizers limit the selection on the basis of a criterion of directly related tables. In a star schema, the fact table is the only table directly related to most other tables, meaning that the fact table is a natural candidate for the first pairwise join. Unfortunately, the fact table is the very largest table in the query, so this strategy leads to selecting a pairwise join order that generates a very large intermediate result set, severely affecting query performance.

There is an optimization strategy, typically referred to as Cartesian Joins, that lessens the performance impact of the pairwise join problem by allowing joining of unrelated tables. The join to the fact table, which is the largest one, is deferred until the very end, thus reducing the size of intermediate result sets. In a join of two unrelated tables every combination of the two tables' rows is produced, a Cartesian product. Such a Cartesian product improves query performance. However, this strategy is viable only if the Cartesian product of dimension rows selected is much smaller than the number of rows in the fact table. The multiplicative nature of the Cartesian join makes the optimization helpful only for relatively small databases.

In addition, systems that exploit hardware and software parallelism have been developed that lessens the performance issues set forth above. Parallelism can help reduce the execution time of a single query (speed-up), or handle additional work without degrading execution time (scale-up).). For example, Red Brick™ has developed STARjoin™ technology that provides high speed, parallelizable multi-table joins in a single pass, thus allowing more than two tables can be joined in a single operation. The core technology is an innovative approach to indexing that accelerates multiple joins. Unfortunately, parallelism can only reduce, not eliminate, the performance degradation issues related to the star schema.

One of the most fundamental principles of the multidimensional database is the idea of aggregation. The most common aggregation is called a roll-up aggregation. This type is relatively easy to compute: e.g. taking daily sales totals and rolling them up into a monthly sales table. The more difficult are analytical calculations, the aggregation of Boolean and comparative operators. However these are also considered as a subset of aggregation.

In a star schema, the results of aggregation are summary tables. Typically, summary tables are generated by database administrators who attempt to anticipate the data aggregations that the users will request, and then pre-build such tables. In such systems, when processing a user-generated query that involves aggregation operations, the pre-built aggregated data that matches the query is retrieved from the summary tables (if such data exists). FIGS. 18A and 18B illustrate a multi-dimensional relational database using a star

schema and summary tables. In this example, the summary tables are generated over the "time" dimension storing aggregated data for "month", "quarter" and "year" time periods as shown in FIG. 18B. Summary tables are in essence additional fact tables, of higher levels. They are attached to the basic fact table creating a snowflake extension of the star schema. There are hierarchies among summary tables because users at different levels of management require different levels of summarization. Choosing the level of aggregation is accomplished via the "drill-down" feature.

Summary tables containing pre-aggregated results typically provide for improved query response time with respect to on-the-fly aggregation. However, summary tables suffer from some disadvantages:

- summary tables require that database administrators anticipate the data aggregation operations that users will require; this is a difficult task in large multi-dimensional databases (for example, in data warehouses and data mining systems), where users always need to query in new ways looking for new information and patterns.
- summary tables do not provide a mechanism that allows efficient drill down to view the raw data that makes up the summary table - typically a table scan of one or more large tables is required.
- querying is delayed until pre-aggregation is completed.
- there is a heavy time overhead because the vast majority of the generated information remains unvisited.
- there is a need to synchronize the summary tables before the use.
- the degree of viable parallelism is limited because the subsequent levels of summary tables must be performed in pipeline, due to their hierarchies.
- for very large databases, this option is not valid because of time and storage space.

Note that it is common to utilize both pre-aggregated results and on-the-fly aggregation in support aggregation. In these system, partial pre-aggregation of the facts results in a small set of summary tables. On-the-fly aggregation is used in the case the required aggregated data does not exist in the summary tables.

Note that in the event that the aggregated data does not exist in the summary tables, table join operations and aggregation operations are performed over the raw facts in order to generate such aggregated data. This is typically referred to as on-the-fly aggregation. In such instances, aggregation indexing is used to mitigate the performance of multiple data joins associated with dynamic aggregation of the raw data. Thus, in large multi-dimensional databases, such dynamic aggregation may lead to unacceptable query response times.

In view of the problems associated with joining and aggregation within RDBMS, prior art ROLAP systems have suffered from essentially the same shortcomings and drawbacks of their underlying RDBMS.

While prior art MOLAP systems provide for improved access time to aggregated data within their underlying MDD structures, and have performance advantages when carrying out joining and aggregations operations, prior art MOLAP architectures have suffered from a

number of shortcomings and drawbacks. More specifically, atomic (raw) data is moved, in a single transfer, to the MOLAP system for aggregation, analysis and querying. Importantly, the aggregation results are external to the DBMS. Thus, users of the DBMS cannot directly view these results. Such results are accessible only from the MOLAP system. Because the MDD query processing logic in prior art MOLAP systems is separate from that of the DBMS, users must procure rights to access to the MOLAP system and be instructed (and be careful to conform to such instructions) to access the MDD (or the DBMS) under certain conditions. Such requirements can present security issues, highly undesirable for system administration. Satisfying such requirements is a costly and logistically cumbersome process. As a result, the widespread applicability of MOLAP systems has been limited.

Thus, there is a great need in the art for an improved mechanism for joining and aggregating data elements within a database management system (e.g., RDBMS), and for integrating the improved database management system (e.g., RDBMS) into informational database systems (including the data warehouse and OLAP domains), while avoiding the shortcomings and drawbacks of prior art systems and methodologies.

SUMMARY AND OBJECTS OF PRESENT INVENTION

Accordingly, it is a further object of the present invention to provide an improved method of and system for managing data elements within a multidimensional database (MDDB) using a novel stand-alone (i.e. external) data aggregation server, achieving a significant increase in system performance (e.g. deceased access/search time) using a stand-alone scalable data aggregation server.

Another object of the present invention is to provide such system, wherein the stand-alone aggregation server includes an aggregation engine that is integrated with an MDDB, to provide a cartridge-style plug-in accelerator which can communicate with virtually any conventional OLAP server.

Another object of the present invention is to provide such a stand-alone data aggregation server whose computational tasks are restricted to data aggregation, leaving all other OLAP functions to the MOLAP server and therefore complementing OLAP server's functionality.

Another object of the present invention is to provide such a system, wherein the stand-alone aggregation server carries out an improved method of data aggregation within the MDDB which enables the dimensions of the MDDB to be scaled up to large numbers and large atomic (i.e. base) data sets to be handled within the MDDB.

Another object of the present invention is to provide such a stand-alone aggregation server, wherein the aggregation engine supports high-performance aggregation (i.e. data roll-up) processes to maximize query performance of large data volumes, and to reduce the time of partial aggregations that degrades the query response.

Another object of the present invention is to provide such a stand-alone, external scalable aggregation server, wherein its integrated data aggregation (i.e. roll-up) engine speeds

up the aggregation process by orders of magnitude, enabling larger database analysis by lowering the aggregation times.

Another object of the present invention is to provide such a novel stand-alone scalable aggregation server for use in OLAP operations, wherein the scalability of the aggregation server enables (i) the speed of the aggregation process carried out therewithin to be substantially increased by distributing the computationally intensive tasks associated with data aggregation among multiple processors, and (ii) the large data sets contained within the MDDB of the aggregation server to be subdivided among multiple processors thus allowing the size of atomic (i.e. basic) data sets within the MDDB to be substantially increased.

Another object of the present invention is to provide such a novel stand-alone scalable aggregation server, which provides for uniform load balancing among processors for high efficiency and best performance, and linear scalability for extending the limits by adding processors.

Another object of the present invention is to provide a stand-alone, external scalable aggregation server, which is suitable for MOLAP as well as for ROLAP system architectures.

Another object of the present invention is to provide a novel stand-alone scalable aggregation server, wherein an MDDB and aggregation engine are integrated and the aggregation engine carries out a high-performance aggregation algorithm and novel storing and searching methods within the MDDB.

Another object of the present invention is to provide a novel stand-alone scalable aggregation server which can be supported on single-processor (i.e. sequential or serial) computing platforms, as well as on multi-processor (i.e. parallel) computing platforms.

Another object of the present invention is to provide a novel stand-alone scalable aggregation server which can be used as a complementary aggregation plug-in to existing MOLAP and ROLAP databases.

Another object of the present invention is to provide a novel stand-alone scalable aggregation server which carries out an novel rollup (i.e. down-up) and spread down (i.e. top-down) aggregation algorithms.

Another object of the present invention is to provide a novel stand-alone scalable aggregation server which includes an integrated MDDB and aggregation engine which carries out full pre-aggregation and/or "on-the-fly" aggregation processes within the MDDB.

Another object of the present invention is to provide such a novel stand-alone scalable aggregation server which is capable of supporting MDDB having a multi-hierarchy dimensionality.

Another object of the present invention is to provide a novel method of aggregating multidimensional data of atomic data sets originating from a RDBMS Data Warehouse.

Another object of the present invention is to provide a novel method of aggregating multidimensional data of atomic data sets originating from other sources, such as external ASCII files, MOLAP server, or other end user applications.

Another object of the present invention is to provide a novel stand-alone scalable data aggregation server which can communicate with any MOLAP server via standard ODBC, OLE DB or DLL interface, in a completely transparent manner with respect to the (client) user, without any time delays in queries, equivalent to storage in MOLAP server's cache.

Another object of the present invention is to provide a novel "cartridge-style" (stand-alone) scalable data aggregation engine which dramatically expands the boundaries of MOLAP into large-scale applications including Banking, Insurance, Retail and Promotion Analysis.

Another object of the present invention is to provide a novel "cartridge-style" (stand-alone) scalable data aggregation engine which dramatically expands the boundaries of high-volatility type ROLAP applications such as, for example, the precalculation of data to maximize query performance.

Another object of the present invention is to provide a generic plug-in cartridge-type data aggregation component, suitable for all MOLAP systems of different vendors, dramatically reducing their aggregation burdens.

Another object of the present invention is to provide a novel high performance cartridge-type data aggregration server which, having standardized interfaces, can be plugged-into the OLAP system of virtually any user or vendor.

Another object of the present invention is to provide a novel "cartridge-style" (stand-alone) scalable data aggregation engine which has the capacity to convert long batch-type data aggregations into interactive sessions.

In another aspect, it is an object of the present invention to provide an improved method of and system for joining and aggregating data elements integrated within a database management system (DBMS) using a non-relational multi-dimensional data structure (MDDB), achieving a significant increase in system performance (e.g. deceased access/search time), user flexibility and ease of use.

Another object of the present invention is to provide such an DBMS wherein its integrated data aggregation module supports high-performance aggregation (i.e. data roll-up) processes to maximize query performance of large data volumes.

Another object of the present invention is to provide such an DBMS system, wherein its integrated data aggregation (i.e. roll-up) module speeds up the aggregation process by orders of magnitude, enabling larger database analysis by lowering the aggregation times.

Another object of the present invention is to provide such a novel DBMS system for use in OLAP operations.

Another object of the present invention is to provide a novel DBMS system having an integrated aggregation module that carries out an novel rollup (i.e. down-up) and spread down (i.e. top-down) aggregation algorithms.

Another object of the present invention is to provide a novel DBMS system having an integrated aggregation module that carries out full pre-aggregation and/or "on-the-fly" aggregation processes.

Another object of the present invention is to provide a novel DBMS system having an integrated aggregation module which is capable of supporting a MDDB having a multi-hierarchy dimensionality.

These and other object of the present invention will become apparent hereinafter and in the Claims to Invention set forth herein.

BRIEF DESCRIPTION OF THE DRAWINGS

In order to more fully appreciate the objects of the present invention, the following Detailed Description of the Illustrative Embodiments should be read in conjunction with the accompanying Drawings, wherein:

Fig. 1A is a schematic representation of an exemplary prior art relational on-line analytical processing (ROLAP) system comprising a three-tier or layer client/server architecture, wherein the first tier has a database layer utilizing an RDBMS for data storage, access, and retrieval processes, the second tier has an application logic layer (i.e. the ROLAP engine) for executing the multidimensional reports from multiple users, and the third tier integrates the ROLAP engine with a variety of presentation layers, through which users perform OLAP analyses;

Fig. 1B is a schematic representation of a generalized embodiment of a prior art multidimensional on-line analytical processing (MOLAP) system comprising a base data loader for receiving atomic (i.e. base) data from a Data Warehouse realized by a RDBMS, an OLAP multidimensional database (MDDB), an aggregation, access and retrival module, application logic module and presentation module associated with a conventional OLAP sever (e.g. Oracle's Express Server) for supporting on-line transactional processing (OLTP) operations on the MDDB, to service database queries and requests from a plurality of OLAP client machines typically accessing the system from an information network (e.g. the Internet);

Fig. 2A is a schematic representation of the Data Warehouse shown in the prior art system of Fig. 1B comprising numerous data tables (e.g. T1, T2, ...Tn) and data field links, and the OLAP multidimensional database shown of Fig. 1B, comprising a conventional page allocation table (PAT) with pointers pointing to the physical storage of variables in an information storage device;

Fig. 2B is a schematic representation of an exemplary three-dimensional MDDB and organized as a 3-dimensional Cartesian cube and used in the prior art system of Fig. 2A, wherein the first dimension of the MDDB is representative of geography (e.g. cities, states, countries, continents), the second dimension of the MDDB is representative of time (e.g. days, weeks, months, years), the third dimension of the MDDB is representative of products

(e.g. all products, by manufacturer), and the basic data element is a set of variables which are addressed by 3-dimensional coordinate values;

Fig. 2C is a schematic representation of a prior art array structure associated with an exemplary three-dimensional MDDB, arranged according to a dimensional hierarchy;

Fig. 2D is a schematic representation of a prior art page allocation table for an exemplary three-dimensional MDDB, arranged according to pages of data element addresses;

Fig. 3A is a schematic representation of a prior art MOLAP system, illustrating the process of periodically storing raw data in the RDBMS Data Warehouse thereof, serially loading of basic data from the Data Warehouse to the MDDB, and the process of serially pre-aggregating (or pre-compiling) the data in the MDDB along the entire dimensional hierarchy thereof;

Fig. 3B is a schematic representation illustrating that the Cartesian addresses listed in a prior art page allocation table (PAT) point to where physical storage of data elements (i.e. variables) occurs in the information recording media (e.g. storage volumes) associated with the MDDB, during the loading of basic data into the MDDB as well as during data preaggregation processes carried out therewithin;

Fig. 3C1 is a schematic representation of an exemplary three-dimensional database used in a conventional MOLAP system of the prior art, showing that each data element contained therein is physically stored at a location in the recording media of the system which is specified by the dimensions (and subdimensions within the dimensional hierarchy) of the data variables which are assigned integer-based coordinates in the MDDB, and also that data elements associated with the basic data loaded into the MDDB are assigned lower integer coordinates in MDDB Space than pre-aggregated data elements contained therewithin;

Fig. 3C2 is a schematic representation illustrating that a conventional hierarchy of the dimension of "time" typically contains the subdimensions "days, weeks, months, quarters, etc." of the prior art;

Fig. 3C3 is a schematic representation showing how data elements having higher subdimensions of time in the MDDB of the prior art are typically assigned increased integer addresses along the time dimension thereof;

Fig. 4 is a schematic representation illustrating that, for very large prior art MDDBs, very large page allocation tables (PATs) are required to represent the address locations of the data elements contained therein, and thus there is a need to employ address data paging techniques between the DRAM (e.g. program memory) and mass storage devices (e.g. recording discs or RAIDs) available on the serial computing platform used to implement such prior art MOLAP systems;

Fig. 5 is a graphical representation showing how search time in a conventional (i.e. prior art) MDDB increases in proportion to the amount of preaggregation of data therewithin;

Fig. 6A is a schematic representation of a generalized embodiment of a multidimensional on-line analytical processing (MOLAP) system of the present invention comprising a Data Warehouse realized as a relational database, a stand-alone Aggregration Server of the present invention having an integrated aggregation engine and MDDB, and an OLAP server supporting a plurality of OLAP clients, wherein the stand-alone Aggregation Server performs aggregation functions (e.g. summation of numbers, as well as other mathematical operations, such as multiplication, subtraction, division etc.) and multi-dimensional data storage functions;

Fig. 6B is a schematic block diagram of the stand-alone Aggregation Server of the illustrative embodiment shown in Fig. 6A, showing its primary components, namely, a base data interface (e.g. OLDB, OLE-DB, ODBC, SQL, JDBC, API, etc.) for receiving RDBMS flat files lists and other files from the Data Warehouse (RDBMS), a base data loader for receiving base data from the base data interface, configuration manager for managing the operation of the base data interface and base data loader, an aggregation engine and MDDB handler for receiving base data from the base loader, performing aggregation operations on the base data, and storing the base data and aggregated data in the MDDB; an aggregation client interface (e.g. OLDB, OLE-DB, ODBC, SQL, JDBC, API, etc.) and input analyzer for receiving requests from OLAP client machines, cooperating with the aggregation engine and MDDB handler to generate aggregated data and/or retrieve aggregated data from the MDDB that pertains to the received requests, and returning this aggregated back to the requesting OLAP clients; and a configuration manager for managing the operation of the input analyzer and the aggregation client interface.

Fig. 6C is a schematic representation of the software modules comprising the aggregation engine and MDDB handler of the stand-alone Aggregation Server of the illustrative embodiment of the present invention, showing a base data list structure being supplied to a hierarchy analysis and reorder module, the output thereof being transferred to an aggregation management module, the output thereof being transferred to a storage module via a storage management module, and a Query Directed Roll-up (QDR) aggregation management module being provided for receiving database (DB) requests from OLAP client machines (via the aggregation client interface) and managing the operation of the aggregation and storage management modules of the present invention;

Fig. 6D is a flow chart representation of the primary operations carried out by the (DB) request serving mechanism within the QDR aggregation management module shown in Fig. 6C;

Fig. 7A is a schematic representation of a separate-platform type implementation of the stand-alone Aggregation Server of the illustrative embodiment of Fig. 6B and a conventional OLAP server supporting a plurality of client machines, wherein base data from a Data Warehouse is shown being received by the aggregation server, realized on a first hardware/software platform (i.e. Platform A) and the stand-alone Aggregation Server is shown serving the conventional OLAP server, realized on a second hardware/software platform (i.e.

Platform B), as well as serving data aggregation requirements of other clients supporting diverse applications such as spreadsheet, GUI front end, and applications;

Fig. 7B is a schematic representation of a shared-platform type implementation of the stand-alone Aggregation Server of the illustrative embodiment of Fig. 6B and a conventional OLAP server supporting a plurality of client machines, wherein base data from a Data Warehouse is shown being received by the stand-alone Aggregation Server, realized on a common hardware/software platform and the aggregation server is shown serving the conventional OLAP server, realized on the same common hardware/software platform, as well as serving data aggregation requirements of other clients supporting diverse applications such as spreadsheet, GUI front end, and applications;

Fig. 8A is a data table setting forth information representative of performance benchmarks obtained by the shared-platform type implementation of the stand-alone Aggregation Server of the illustrative embodiment serving the conventional OLAP server (i.e. Oracle EXPRESS Server) shown in Fig. 7B, wherein the common hardware/software platform is realized using a Pentium II 450Mhz, 1GB RAM, 18GB Disk, running the Microsoft NT operating system (OS);

Fig. 9A is a schematic representation of the first stage in the method of segmented aggregation according to the principles of the present invention, showing initial aggregration along the 1st dimension;

Fig. 9B is a schematic representation of the next stage in the method of segmented aggregation according to the principles of the present invention, showing that any segment along dimension 1, such as the shown slice, can be separately aggregated along the remaining dimensions, 2 and 3, and that in general, for an N dimensional system, the second stage involves aggregation in N-1 dimensions. The principle of segementation can be applied on the first stage as well, however, only a large enough data will justify such a sliced procedure in the first dimension. Actually, it is possible to consider each segment as an N-1 cube, enabling recursive computation.

Fig. 9C1 is a schematic representation of the Query Directed Roll-up (QDR) aggregation method/procedure of the present invention, showing data aggregation starting from existing basic data or previously aggregated data in the first dimension (D1), and such aggregated data being utilized as a basis for QDR aggregation along the second dimension (D2);

Fig. 9C2 is a schematic representation of the Query Directed Roll-up (QDR) aggregation method/procedure of the present invention, showing initial data aggregation starting from existing previously aggregated data in the second third (D3), and continuing along the third dimension (D3), and thereafter continuing aggregation along the second dimension (D2);

Fig. 10A is a schematic representation of the "slice-storage" method of storing sparse data in the disk storage devices of the MDDB of Fig. 6B in accordance with the principles of the present invention, based on an ascending-ordered index along aggregation direction, enabling fast retrieval of data;

Fig. 10B is a schematic representation of the data organization of data files and the directory file used in the storages of the MDDB of Fig. 6B, and the method of searching for a queried data point therein using a simple binary search technique due to the data files ascending order;

Fig. 11A is a schematic representation of three exemplary multi-hierarchical data structures for storage of data within the MDDB of Fig. 6B, having three levels of hierarchy, wherein the first level representative of base data is composed of items A,B,F, and G, the second level is composed of items C,E,H and I, and the third level is composed of a single item D, which is common to all three hierarchical structures;

Fig. 11B is a schematic representation of an optimized multi-hierarchical data structure merged from all three hierarchies of Fig. 11A, in accordance with the principles of the present invention;

Fig. 11C(i) through 11C(ix) represent a flow chart description (and accompanying data structures) of the operations of an exemplary hierarchy transformation mechanism of the present invention that optimally merges multiple hierarchies into a single hierarchy that is functionally equivalent to the multiple hierarchies.

Fig. 12 is a schematic representation showing the levels of operations performed by the stand-alone Aggregation Server of Fig. 6B, summarizing the different enabling components for carrying out the method of segmented aggregation in accordance with the principles of the present invention;

Fig. 13 is a schematic representation of the stand-alone Aggregation Server of the present invention shown as a component of a central data warehouse, serving the data aggregation needs of URL directory systems, Data Marts, RDBMSs, ROLAP systems and OLAP systems alike;

Fig. 14 is a schematic representation of a prior art information database system, wherein the present invention may be embodied;

Fig. 15 is a schematic representation of the prior art data warehouse and OLAP system, wherein the present invention may be embodied;

Figs. 16A-16C are schematic representations of exemplary tables employed in a prior art Relational Database Management System (RDBMS); Figs. 16B and 16C illustrate operators (queries) on the table of Fig. 16A, and the result of such queries, respectively;

Fig. 17A is a schematic representation of an exemplary dimensional schema (star schema) of a relational database;

Fig. 17B is a schematic representation of tables used to implement the schema shown in Fig. 17A;

Fig. 18A is a schematic representation of an exemplary multidimensional schema (star schema);

Fig. 18B is a schematic representation of tables used to implement the schema of Fig. 18A, including summary tables storing results of aggregation operations performed on the facts of the central fact table along the time-period dimension, in accordance with conventional teachings;

Fig. 19A is a schematic representation of an exemplary embodiment of a DBMS (for example, an RDBMS as shown) of the present invention comprising a relational datastore and an integrated multidimensional (MDD) aggregation module supporting queries from a plurality of clients, wherein the aggregation engine performs aggregation functions (e.g. summation of numbers, as well as other mathematical operations, such as multiplication, subtraction, division etc.) and non-relational multi-dimensional data storage functions.

Fig. 19B is a schematic block diagram of the MDD aggregation module of the illustrative embodiment of the present invention shown in Fig. 6A.

Fig. 19C(i) and 19C(ii), taken together, set forth a flow chart representation of the primary operations carried out within the DBMS of the present invention when performing data aggregation and related support operations, including the servicing of user-submitted (e.g. natural language) queries made on such aggregated database of the present invention.

Fig. 19D is a flow chart representation of the primary operations carried out by the (DB) request serving mechanism within the MDD control module shown in Fig. 6B.

Fig. 19E is a schematic representation of the view mechanism of an DBMS that enables users to query on the aggregated data generated and/or stored in the MDD Aggregation module according to the present invention.

Fig. 19F is a schematic representation of the trigger mechanism of the DBMS that enables users to query on the aggregated data generated and/or stored in the MDD Aggregation module according to the present invention.

Fig. 19G is a schematic representation of the DBMS of the present invention, illustrating a logically partitioning into a relational part and a non-relational part. The relational part includes the relational data store (e.g., table(s) and dictionary) and support mechanisms (e.g., query handling services). The non-relational part includes the MDD Aggregation Module. Data flows bidirectionally between the relational part and the non-relational part as shown.

Fig. 20A shows a separate-platform type implementation of the DBMS system of the illustrative embodiment shown in Fig. 19A, wherein the relational datastore and support mechanisms (e.g., query handling, fact table(s) and dictionary of the DBMS) reside on a separate hardware platform and/or OS system from that used to run the MDD Aggregation Module of the present invention.

Fig. 20B shows a common-platform type implementation of the DBMS system of the illustrative embodiment shown in Fig. 19A, wherein the relational datastore and support mechanisms (e.g., query handling, fact table(s) and dictionary of the DBMS) share the same hardware platform and operating system (OS) that is used to run the MDD Aggregation Module of the present invention.

Fig. 21 is a schematic representation of the DBMS of the present invention shown as a component of a central data warehouse, serving the data storage and aggregation needs of a ROLAP system (or other OLAP system).

Fig. 22 is a schematic representation of the DBMS of the present invention shown as a component of a central data warehouse, wherein the DBMS includes integrated OLAP

Analysis Logic (and preferably an integrated Presentation Module) that operates cooperatively with the query handling of the DBMS system and the MDD Aggregation Module to enable users of the DBMS system to execute multidimensional reports (e.g., ratios, ranks, transforms, dynamic consolidation, complex filtering, forecasts, query governing, scheduling, flow control, pre-aggregate inferencing, denomalization support, and/or table partitioning and joins) and preferably perform traditional OLAP analyses (grids, graphs, maps, alerts, drill-down, data pivot, data surf, slice and dice, print).

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS OF THE PRESENT INVENTION

Referring now to Figs. 6A through 13, the preferred embodiments of the method and system of the present invention will be now described in great detail hereinbelow, wherein like elements in the Drawings shall be indicated by like reference numerals.

Through this invention disclosure, the term "aggregation" and "preaggregation" shall be understood to mean the process of summation of numbers, as well as other mathematical operations, such as multiplication, subtraction, division etc.

In general, the stand-alone aggregation server and methods of and apparatus for data aggregation of the present invention can be employed in a wide range of applications, including MOLAP systems, ROLAP systems, Internet URL-directory systems, personalized on-line e-commerce shopping systems, Internet-based systems requiring real-time control of packet routing and/or switching, and the like.

For purposes of illustration, initial focus will be accorded to improvements in MOLAP systems, in which knowledge workers are enabled to intuitively, quickly, and flexibly manipulate operational data within a MDDB using familiar business terms in order to provide analytical insight into a business domain of interest.

Fig. 6A illustrates a generalized embodiment of a multidimensional on-line analytical processing (MOLAP) system of the present invention comprising: a Data Warehouse realized as a relational database; a stand-alone cartridge-style Aggregation Server of the present invention having an integrated aggregation engine and a MDDB; and an OLAP server communicating with the Aggregation Server, and supporting a plurality of OLAP clients. In accordance with the principles of the present invention, the stand-alone Aggregation Server performs aggregation functions (e.g. summation of numbers, as well as other mathematical operations, such as multiplication, subtraction, division etc.) and multi-dimensional data storage functions.

Departing from conventional practices, the principles of the present invention teaches moving the aggregation engine and the MDDB into a separate Aggregation Server having standardized interfaces so that it can be plugged-into the OLAP server of virtually any user or vendor. This dramatic move discontinues the restricting dependency of aggregation from the analytical functions of OLAP, and by applying novel and independent algorithms. The stand-

alone data aggregation server enables efficient organization and handling of data, fast aggregation processing, and fast access to and retrieval of any data element in the MDDB.

As will be described in greater detail hereinafter, the Aggregation Server of the present invention can serve the data aggregation requirements of other types of systems besides OLAP systems such as, for example, URL directory management Data Marts, RDBMS, or ROLAP systems.

The Aggregation Server of the present invention excels in performing two distinct functions, namely: the aggregation of data in the MDDB; and the handling of the resulting data base in the MDDB, for "on demand" client use. In the case of serving an OLAP server, the Aggregation Server of the present invention focuses on performing these two functions in a high performance manner (i.e. aggregating and storing base data, originated at the Data Warehouse, in a multidimensional storage (MDDB), and providing the results of this data aggregation process "on demand" to the clients, such as the OLAP server, spreadsheet applications, the end user applications. As such, the Aggregation Server of the present invention frees each conventional OLAP server, with which it interfaces, from the need of making data aggregations, and therefore allows the conventional OLAP server to concentrate on the primary functions of OLAP servers, namely: data analysis and supporting a graphical interface with the user client.

Fig. 6B shows the primary components of the stand-alone Aggregation Server of the illustrative embodiment, namely: a base data interface (e.g. OLDB, OLE-DB, ODBC, SQL, JDBC, API, etc.) for receiving RDBMS flat files lists and other files from the Data Warehouse (RDBMS), a base data loader for receiving base data from the base data interface, configuration manager for managing the operation of the base data interface and base data loader, an aggregation engine for receiving base data from the base loader, a multi-dimensional database (MDDB); a MDDB handler, an input analyzer, an aggregation client interface (e.g. OLDB, OLE-DB, ODBC, SQL, API, JDBC, etc.) and a configuration manager for managing the operation of the input analyzer and the aggregation client interface.

During operation, the base data originates at data warehouse or other sources, such as external ASCII files, MOLAP server, or others. The Configuration Manager, in order to enable proper communication with all possible sources and data structures, configures two blocks, the Base Data Interface and Data Loader. Their configuration is matched with different standards such as OLDB, OLE-DB, ODBC, SQL, API, JDBC, etc.

As shown in Fig. 6B, the core of the data Aggregation Server of the present invention comprises: a data Aggregation Engine; a Multidimensional Data Handler (MDDB Handler); and a Multidimensional Data Storage (MDDB). The results of data aggregation are efficiently stored in the MDDB by the MDDB Handler.

As shown in Figs. 6A and 6B, the stand-alone Aggregation Server of the present invention serves the OLAP Server (or other requesting computing system) via an aggregation client interface, which preferably conforms to standard interface protocols such as OLDB, OLE-DB, ODBC, SQL, JDBC, an API, etc. Aggregation results required by the OLAP server are supplied on demand. Typically, the OLAP Server disintegrates the query, via parsing

process, into series of requests. Each such request, specifying a n-dimensional coordinate, is presented to the Aggregation Server. The Configuration Manager sets the Aggregation Client Interface and Input Analyzer for a proper communication protocol according to the client user. The Input Analyzer converts the input format to make it suitable for the MDDB Handler.

An object of the present invention is to make the transfer of data completely transparent to the OLAP user, in a manner which is equivalent to the storing of data in the MOLAP server's cache and without any query delays. This requires that the stand-alone Aggregation Server have exceptionally fast response characteristics. This object is enabled by providing the unique data structure and aggregation mechanism of the present invention.

Fig. 6C shows the software modules comprising the aggregation engine and MDDB handler components of the stand-alone Aggregation Server of the illustrative embodiment. The base data list, as it arrives from RDBMS or text files, has to be analyzed and reordered to optimize hierarchy handling, according to the unique method of the present invention, as described later with reference to Figs. 11A and 11B.

The function of the aggregation management module is to administrate the aggregation process according to the method illustrated in Figs. 9A and 9B.

In accordance with the principles of the present invention, data aggregation within the stand-alone Aggregation Server can be carried out either as a complete pre-aggregation process, where the base data is fully aggregated before commencing querying, or as a query directed roll-up (QDR) process, where querying is allowed at any stage of aggregation using the "on-the-fly" data aggregation process of the present invention. The QDR process will be described hereinafter in greater detail with reference to Fig. 9C. The response to a request (i.e. a basic component of a client query), by calling the Aggregation management module for "on-the-fly" data aggregation, or for accessing pre-aggregated result data via the Storage management module. The query/request serving mechanism of the present invention within the QDR aggregation management module is illustrated in the flow chart of Fig. 6D.

The function of the Storage management module is to handle multidimensional data in the storage(s) module in a very efficient way, according to the novel method of the present invention, which will be described in detail hereinafter with reference to Figs. 10A and 10B.

The request serving mechanism shown in Fig. 6D is controlled by the QDR aggregation management module. Requests are queued and served one by one. If the required data is already pre-calculated, then it is retrieved by the storage management module and returned to the client. Otherwise, the required data is calculated "on-the-fly" by the aggregation management module, and the result moved out to the client, while simultaneously stored by the storage management module, shown in Fig. 6C.

Figs. 7A and 7B outline two different implementations of the stand-alone (cartridge-style) Aggregation Server of the present invention. In both implementations, the Aggregation Server supplies aggregated results to a client.

Fig. 7A shows a separate-platform type implementation of the MOLAP system of the illustrative embodiment shown in Fig. 6A, wherein the Aggregation Server of the present invention resides on a separate hardware platform and OS system from that used to run the OLAP server. In this type of implementation, it is even possible to run the Aggregation Server and the OLAP Server on different-type operating systems (e.g. NT, Unix, MAC OS).

Fig. 7B shows a common-platform type implementation of the MOLAP system of the illustrative embodiment shown in Fig. 6B, wherein the Aggregation Server of the present invention and OLAP Server share the same hardware platform and operating system (OS).

Fig. 8A shows a table setting forth the benchmark results of an aggregation engine, implemented on a shared/common hardware platform and OS, in accordance with the principles of the present invention. The common platform and OS is realized using a Pentium II 450Mhz, 1GB RAM, 18GB Disk, running the Microsoft NT operating system. The six (6) data sets shown in the table differ in number of dimensions, number of hierarchies, measure of sparcity and data size. A comparison with ORACLE Express, a major OLAP server, is made. It is evident that the aggregation engine of the present invention outperforms currently leading aggregation technology by more than an order of magnitude.

The segmented data aggregation method of the present invention is described in Figs. 9A through 9C2. These figures outline a simplified setting of three dimensions only; however, the following analysis applies to any number of dimensions as well.

The data is being divided into autonomic segments to minimize the amount of simultaneously handled data. The initial aggregation is practiced on a single dimension only, while later on the aggregation process involves all other dimensions.

At the first stage of the aggregation method, an aggregation is performed along dimension 1. The first stage can be performed on more than one dimension. As shown in Fig. 9A, the space of the base data is expanded by the aggregation process.

In the next stage shown in Fig. 9B, any segment along dimension 1, such as the shown slice, can be separately aggregated along the remaining dimensions, 2 and 3. In general, for an N dimensional system, the second stage involves aggregation in N-1 dimensions.

The principle of data segmentation can be applied on the first stage as well. However, only a large enough data set will justify such a sliced procedure in the first dimension. Actually, it is possible to consider each segment as an N-1 cube, enabling recursive computation.

It is imperative to get aggregation results of a specific slice before the entire aggregation is completed, or alternatively, to have the roll-up done in a particular sequence. This novel feature of the aggregation method of the present invention is that it allows the querying to begin, even before the regular aggregation process is accomplished, and still having fast response. Moreover, in relational OLAP and other systems requiring only partial aggregations, the QDR process dramatically speeds up the query response.

The QDR process is made feasible by the slice-oriented roll-up method of the present invention. After aggregating the first dimension(s), the multidimensional space is composed

of independent multidimensional cubes (slices). These cubes can be processed in any arbitrary sequence.

Consequently the aggregation process of the present invention can be monitored by means of files, shared memory sockets, or queues to statically or dynamically set the roll-up order.

In order to satisfy a single query coming from a client, before the required aggregation result has been prepared, the QDR process of the present invention involves performing a fast on-the-fly aggregation (roll-up) involving only a thin slice of the multidimensional data.

Fig. 9C1 shows a slice required for building-up a roll-up result of the 2nd dimension. In case 1, as shown, the aggregation starts from an existing data, either basic or previously aggregated in the first dimension. This data is utilized as a basis for QDR aggregation along the second dimension. In case 2, due to lack of previous data, a QDR involves an initial slice aggregation along dimension 3, and thereafter aggregation along the 2nd dimension.

Fig. 9C2 shows two corresponding QDR cases for gaining results in the 3d dimension. Cases 1 and 2 differ in the amount of initial aggregation required in 2nd dimension.

Fig. 10A illustrates the "Slice-Storage" method of storing sparse data on storage disks. In general, this data storage method is based on the principle that an ascending-ordered index along aggregation direction, enables fast retrieval of data. Fig. 10A illustrates a unit-wide slice of the multidimensional cube of data. Since the data is sparse, only few non-NA data points exist. These points are indexed as follows. The Data File consists of data records, in which each *n-1* dimensional slice is being stored, in a separate record. These records have a varying length, according to the amount of non-NA stored points. For each registered point in the record, IND_k stands for an index in a n-dimensional cube, and *Data* stands for the value of a given point in the cube.

Fig. 10B illustrates a novel method for randomly searching for a queried data point in the MDDB of Fig. 6B by using a novel technique of organizing data files and the directory file used in the storages of the MDDB, so that a simple binary search technique can then be employed within the Aggregation Server of the prsent invention. According to this method, a metafile termed DIR File, keeps pointers to Data Files as well as additional parameters such as the start and end addresses of data record (IND_0, IND_n), its location within the Data File, record size (n), file's physical address on disk (*D_Path*), and auxiliary information on the record (*Flags*).

A search for a queried data point is then performed by an access to the DIR file. The search along the file can be made using a simple binary search due to file's ascending order. When the record is found, it is then loaded into main memory to search for the required point, characterized by its index IND_k. The attached *Data* field represents the queried value. In case the exact index is not found, it means that the point is a NA.

In another aspect of the present invention, a novel method is provided for optimally merging multiple hierarchies in multi-hierarchical structures. The method, illustrated in Figs. 11A, 11B, and 11C is preferably used by the Aggregation Server of the present invention in processing the table data (base data), as it arrives from RDBMS.

According to the devised method, the inner order of hierarchies within a dimension is optimized, to achieve efficient data handling for summations and other mathematical formulas (termed in general "Aggregation"). The order of hierarchy is defined externally. It is brought from a data source to the stand-alone aggregation engine, as a descriptor of data, before the data itself. In the illustrative embodiment, the method assumes hierarchical relations of the data, as shown in Fig. 11A. The way data items are ordered in the memory space of the Aggregation Server, with regard to the hierarchy, has a significant impact on its data handling efficiency.

Notably, when using prior art techniques, multiple handling of data elements, which occurs when a data element is accessed more than once during aggregation process, has been hitherto unavoidable when the main concern is to effectively handle the sparse data. The data structures used in prior art data handling methods have been designed for fast access to a non NA data. According to prior art techniques, each access is associated with a timely search and retrieval in the data structure. For the massive amount of data typically accessed from a Data Warehouse in an OLAP application, such multiple handling of data elements has significantly degraded the efficiency of prior art data aggregation processes. When using prior art data handling techniques, the data element *D shown in* Fig. 11A must be accessed three times, causing poor aggregation performance.

In accordance with the data handling method of the present invention, the data is being pre-ordered for a singular handling, as opposed to multiple handling taught by prior art methods. According to the present invention, elements of base data and their aggregated results are contiguously stored in a way that each element will be accessed only once. This particular order allows a forward-only handling, never backward. Once a base data element is stored, or aggregated result is generated and stored, it is never to be retrieved again for further aggregation. As a result the storage access is minimized. This way of singular handling greatly elevates the aggregation efficiency of large data bases. An efficient handling method as used in the present invention, is shown in Fig. 7A. The data element D, as any other element, is accessed and handled only once.

Fig. 11A shows an example of a multi-hierarchical database structure having 3 hierarchies. As shown, the base data has a dimension that includes items A,B,F, and G., The second level is composed of items C,E,H and I. The third level has a single item D, which is common to all three hierarchical structures. In accordance with the method of the present invention, a minimal computing path is always taken. For example, according to the method of the present invention, item D will be calculated as part of structure 1, requiring two mathematical operations only, rather than as in structure 3, which would need four mathematical operations. Fig. 11B depicts an optimized structure merged from all three hierarchies.

Fig. 11C(i) through 11C(ix) represent a flow chart description (and accompanying data structures) of the operations of an exemplary hierarchy transformation mechanism of the present invention that optimally merges multiple hierarchies into a single hierarchy that is functionally equivalent to the multiple hierarchies. For the sake of description, the data

structures correspond to exemplary hierarchical structures described above with respect to Figs. 11(A) and 11(B). As illustrated in Fig. 11C(i), in step 1101, a catalogue is loaded from the DBMS system. As is conventional, the catalogue includes data ("hierarchy descriptor data") describing multiple hierarchies for at least one dimension of the data stored in the DBMS. In step 1103, this hierarchy descriptor data is extracted from the catalogue. A loop (steps 1105-1119) is performed over the items in the multiple hierarchy described by the hierarchy descriptor data.

In the loop 1105-1119, a given item in the multiple hierarchy is selected (step 1107); and, in step 1109, the parent(s) (if any) - including grandparents, great-grandparents, etc. - of the given item are identified and added to an entry (for the given item) in a parent list data structure, which is illustrated in Fig. 11C(v). Each entry in the parent list corresponds to a specific item and includes zero or more identifiers for items that are parents (or grandparents, or great-grandparents) of the specific item. In addition, an inner loop (steps 1111-1117) is performed over the hierarchies of the multiple hierarchies described by the hierarchy descriptor data, wherein in step 1113 one of the multiple hierarchies is selected. In step 1115, the child of the given item in the selected hierarchy (if any) is identified and added (if need be) to a group of identifiers in an entry (for the given item) in a child list data structure, which is illustrated in Fig. 11C(vi). Each entry in the child list corresponds to a specific item and includes zero or more groups of identifiers each identifying a child of the specific item. Each group corresponds to one or more of the hierarchies described by the hierarchy descriptor data.

The operation then continues to steps 1121 and 1123 as illustrated in Fig. 11C(ii) to verify the integrity of the multiple hierarchies described by the hierarchy descriptor data (step 1121) and fix (or report to the user) any errors discovered therein (step 1123). Preferably, the integrity of the multiple hierarchies is verified in step 1121 by iteratively expanding each group of identifiers in the child list to include the children, grandchilren, etc of any item listed in the group. If the child(ren) for each group for a specific item do not match, a verification error is encountered, and such error is fixed (or reported to the user (step 1123). The operation then proceeds to a loop (steps 1125 - 1133) over the items in the child list.

In the loop (steps 1125 - 1133), a given item in the child list is identified in step 1127. In step 1129, the entry in the child list for the given item is examined to determine if the given item has no children (e.g., the corresponding entry is null). If so, the operation continues to step 1131 to add an entry for the item in level 0 of an ordered list data structure, which is illustrated in Fig. 11C(vii); otherwise the operation continues to process the next item of the child list in the loop. Each entry in a given level of the order list corresponds to a specific item and includes zero or more identifiers each identifying a child of the specific item. The levels of the order list described the transformed hierarchy as will readily become apparent in light of the following. Essentially, loop 1125-1333 builds the lowest level (level 0) of the transformed hierarchy.

After loop 1125-1133, operation continues to process the lowest level to derive the next higher level, and iterate over this process to build out the entire transformed hierarchy.

More specifically, in step 1135, a "current level" variable is set to identify the lowest level. In step 1137, the items of the "current level" of the ordered list are copied to a work list. In step 1139, it is determined if the worklist is empty. If so, the operation ends; otherwise operation continues to step 1141 wherein a loop (steps 1141 - 1159) is performed over the items in the work list.

In step 1143, a given item in the work list is identified and operation continues to an inner loop (steps 1145 - 1155) over the parent(s) of the given item (which are specified in the parent list entry for the given item). In step 1147 of the inner loop, a given parent of the given item is identified. In step 1149, it is determined whether any other parent (e.g., a parent other than the given patent) of the given item is a child of the given parent (as specified in the child list entry for the given parent). If so, operation continues to step 1155 to process the next parent of the given item in the inner loop; otherwise, operation continues to steps 1151 and 1153. In step 1151, an entry for the given parent is added to the next level (current level + 1) of the ordered list, if it does not exist there already. In step 1153, if no children of the given item (as specified in the entry for the given item in the current level of the ordered list) matches (e.g., is covered by) any child (or grandchild or great grandchild etc) of item(s) in the entry for the given parent in the next level of the ordered list, the given item is added to the entry for the given parent in the next level of the ordered list. Levels 1 and 2 of the ordered list for the example described above are shown in Figs. 11C(viii) and 11C(ix), respectively. The children (including grandchildren and great grandchildren. etc) of an item in the entry for a given parent in the next level of the ordered list may be identified by the information encoded in the lower levels of the ordered list. After step 1153, operation continues to step 1155 to process the next parent of the given item in the inner loop (steps 1145 - 1155)

After processing the inner loop (steps 1145 - 1155), operation continues to step 1157 to delete the given item from the work list, and processing continues to step 1159 to process the next item of the work list in the loop (steps 1141 - 1159).

After processing the loop (steps 1141 - 1159), the ordered list (e.g., transformed hierarchy) has been built for the next higher level. The operation continues to step 1161 to increment the current level to the next higher level, and operation returns (in step 1163) to step 1138 to build the next higher level , until the highest level is reached (determined in step 1139) and the operation ends.

Fig. 12 summarizes the components of an exemplary aggregation module that takes advantage of the hierarchy transformation technique described above. More specifically, the aggregation module includes an hierarchy transformation module that optimally merges multiple hierarchies into a single hierarchy that is functionally equivalent to the multiple hierarchies. A second module loads and indexes the base data supplied from the DBMS using the optimal hierarchy generated by the hierarchy transformation module. An aggregation engine performs aggregation operations on the base data. During the aggregation operations along the dimension specified by the optimal hierarchy, the results of the aggregation operations of the level 0 items may be used in the aggregation operations of the

level 1 items, the results of the aggregation operations of the level 1 items may be used in the aggregation operations of the level 2 items, etc. Based on these operations, the loading and indexing operations of the base data, along with the aggregation become very efficient, minimizing memory and storage access, and speeding up storing and retrieval operations.

Fig. 13 shows the stand-alone Aggregation Server of the present invention as a component of a central data warehouse, serving the data aggregation needs of URL directory systems, Data Marts, RDBMSs, ROLAP systems and OLAP systems alike.

The reason for the central multidimensional database's rise to corporate necessity is that it facilitates flexible, high-performance access and analysis of large volumes of complex and interrelated data.

A stand-alone specialized aggregation server, simultaneously serving many different kinds of clients (e.g. data mart, OLAP, URL, RDBMS), has the power of delivering an enterprise-wide aggregation in a cost-effective way. This kind of server eliminates the roll-up redundancy over the group of clients, delivering scalability and flexibility.

Performance associated with central data warehouse is an important consideration in the overall approach. Performance includes aggregation times and query response.

Effective interactive query applications require near real-time performance, measured in seconds. These application performances translate directly into the aggregation requirements.

In the prior art, in case of MOLAP, a full pre-aggregation must be done before starting querying. In the present invention, in contrast to prior art, the query directed roll-up (QDR) allows instant querying, while the full pre-aggregation is done in the background. In cases a full pre-aggregation is preferred, the currently invented aggregation outperforms any prior art. For the ROLAP and RDBMS clients, partial aggregations maximize query performance. In both cases fast aggregation process is imperative. The aggregation performance of the current invention is by orders of magnitude higher than that of the prior art.

The stand-alone scalable aggregation server of the present invention can be used in any MOLAP system environment for answering questions about corporate performance in a particular market, economic trends, consumer behaviors, weather conditions, population trends, or the state of any physical, social, biological or other system or phenomenon on which different types or categories of information, organizable in accordance with a predetermined dimensional hierarchy, are collected and stored within a RDBMS of one sort or another. Regardless of the particular application selected, the address data mapping processes of the present invention will provide a quick and efficient way of managing a MDDB and also enabling decision support capabilities utilizing the same in diverse application environments.

The stand-alone "cartridge-style" plug-in features of the data aggregation server of the present invention, provides freedom in designing an optimized multidimensional data structure and handling method for aggregation, provides freedom in designing a generic

aggregation server matching all OLAP vendors, and enables enterprise-wide centralized aggregation.

The method of Segmented Aggregation employed in the aggregation server of the present invention provides flexibility, scalability, a condition for Query Directed Aggregation, and speed improvement.

The method of Multidimensional data organization and indexing employed in the aggregation server of the present invention provides fast storage and retrieval, a condition for Segmented Aggregation, improves the storing, handling, and retrieval of data in a fast manner, and contributes to structural flexibility to allow sliced aggregation and QDR. It also enables the forwarding and single handling of data with improvements in speed performance.

The method of Query Directed Aggregation (QDR) employed in the aggregation server of the present invention minimizes the data handling operations in multi-hierarchy data structures.

The method of Query Directed Aggregation (QDR) employed in the aggregation server of the present invention eliminates the need to wait for full aggregation to be completed, and provides build-up aggregated data required for full aggregation.

In another aspect of the present invention, an improved DBMS system (e.g., RDBMS system , object oriented database system or object/relational database system) is provided that excels in performing two distinct functions, namely: the aggregation of data; and the handling of the resulting data for "on demand" client use. Moreover, because of improved data aggregation capabilities, the DBMS of the present invention can be employed in a wide range of applications, including Data Warehouses supporting OLAP systems and the like. For purposes of illustration, initial focus will be accorded to the DBMS of the present invention. Referring now to Figs. 19 through Figs. 21, the preferred embodiments of the method and system of the present invention will be now described in great detail herein below.

Through this document, the term "aggregation" and "pre-aggregation" shall be understood to mean the process of summation of numbers, as well as other mathematical operations, such as multiplication, subtraction, division etc. It shall be understood that pre-aggregation operations occur asynchronously with respect to the traditional query processing operations. Moreover, the term "atomic data" shall be understood to refer to the lowest level of data granularity required for effective decision making. In the case of a retail merchandising manager, atomic data may refer to information by store, by day, and by item. For a banker, atomic data may be information by account, by transaction, and by branch.

FIG. 19A illustrates the primary components of an illustrative embodiment of an DBMS of the present invention, namely: support mechanisms including a query interface and query handler; a relational data store including one or more tables storing at least the atomic data (and possibly summary tables) and a meta-data store for storing a dictionary (sometimes referred to as a catalogue or directory); and an MDD Aggregation Module that stores atomic data and aggregated data in a MDDB. The MDDB is a non-relational data structure - it uses other data structures, either instead of or in addition to tables - to store data. For illustrative

purposes, Fig. 19A illustrates an RDBMS wherein the relational data store includes fact tables and a dictionary.

It should be noted that the DBMS typically includes additional components (not shown) that are not relevant to the present invention. The query interface and query handler service user-submitted queries (in the preferred embodiment, SQL query statements) forwarded, for example, from a client machine over a network as shown. The query handler and relational data store (tables and meta-data store) are operably coupled to the MDD Aggregation Module. Importantly, the query handler and integrated MDD Aggregation Module operate to provide for dramatically improved query response times for data aggregation operations and drill-downs. Moreover, it is an object of the present invention is to make user-querying of the non-relational MDDB no different than querying a relational table of the DBMS, in a manner that minimizes the delays associated with queries that involve aggregation or drill down operations. This object is enabled by providing the novel DBMS system and integrated aggregation mechanism of the present invention.

FIG. 19B shows the primary components of an illustrative embodiment of the MDD Aggregation Module of FIG. 19A, namely: a base data loader for loading the directory and table(s) of relational data store of the DBMS; an aggregation engine for receiving dimension data and atomic data from the base loader, a multi-dimensional database (MDDB); a MDDB handler and an SQL handler that operate cooperatively with the query handler of the DBMS to provide users with query access to the MDD Aggregation Module, and a control module for managing the operation of the components of the MDD aggregation module. The base data loader may load the directory and table(s) of the relational data store over a standard interface (such as OLDB, OLE-DB, ODBC, SQL, API, JDBC, etc.). In this case, the DBMS and base data loader include components that provide communication of such data over these standard interfaces. Such interface components are well known in the art. For example, such interface components are readily available from Attunity Corporation, http://www.attunity.com.

During operation, base data originates from the table(s) of the DBMS. The core data aggregation operations are performed by the Aggregation Engine; a Multidimensional Data (MDDB) Handler; and a Multidimensional Data Storage (MDDB). The results of data aggregation are efficiently stored in the MDDB by the MDDB Handler. The SQL handler of the MDD Aggregation module services user-submitted queries (in the preferred embodiment, SQL query statements) forwarded from the query handler of the DBMS. The SQL handler of the MDD Aggregation module may communicate with the query handler of the DBMS over a standard interface (such as OLDB, OLE-DB, ODBC, SQL, API, JDBC, etc.). In this case, the support mechanisms of the RDBMS and SQL handler include components that provide communication of such data over these standard interfaces. Such interface components are well known in the art. Aggregation (or drill down results) are retrieved on demand and returned to the user.

Typically, a user interacts with a client machine (for example, using a web-enabled browser) to generate a natural language query, that is communicated to the query interface of

the DBMS, for example over a network as shown. The query interface disintegrates the query, via parsing, into a series of requests (in the preferred embodiment, SQL statements) that are communicated to the query handler of the DBMS. It should be noted that the functions of the query interface may be implemented in a module that is not part of the DBMS (for example, in the client machine). The query handler of the DBMS forwards requests that involve data stored in the MDD of the MDD Aggregation module to the SQL hander of the MDD Aggregation module for servicing. Each request specifies a set of n-dimensions. The SQL handler of the MDD Aggregation Module extracts this set of dimensions and operates cooperatively with the MDD handler to address the MDDB using the set of dimensions, retrieve the addressed data from the MDDB, and return the results to the user via the query handler of the DBMS.

Fig. 19C(i) and 19C(ii) is a flow chart illustrating the operations of an illustrative DBMS of the present invention. In step 601, the base data loader of the MDD Aggregation Module loads the dictionary (or catalog) from the meta-data store of the DBMS. In performing this function, the base data loader may utilize an adapter (interface) that maps the data types of the dictionary of the DBMS (or that maps a standard data type used to represent the dictionary of the DBMS) into the data types used in the MDD aggregation module. In addition, the base data loader extracts the dimensions from the dictionary and forwards the dimensions to the aggregation engine of the MDD Aggregation Module.

In step 603, the base data loader loads table(s) from the DBMS. In performing this function, the base data loader may utilize an adapter (interface) that maps the data types of the table(s) of the DBMS (or that maps a standard data type used to represent the fact table(s) of the DBMS) into the data types used in the MDD Aggregation Module. In addition, the base data loader extracts the atomic data from the table(s), and forwards the atomic data to the aggregation engine.

In step 605, the aggregation engine performs aggregation operations (i.e., roll-up operation) on the atomic data (provided by the base data loader in step 603) along at least one of the dimensions (extracted from the dictionary of the DBMS in step 601) and operates cooperatively with the MDD handler to store the resultant aggregated data in the MDDB. A more detailed description of exemplary aggregation operations according to a preferred embodiment of the present invention is set forth below with respect to the QDR process of Figs. 9A-9C.

In step 607, a reference is defined that provides users with the ability to query the data generated by the MDD Aggregation Module and/or stored in the MDDB of the MDD Aggregation Module. This reference is preferably defined using the Create View SQL statement, which allows the user to: i) define a table name (TN) associated with the MDDB stored in the MDD Aggregation Module, and ii) define a link used to route SQL statements on the table TN to the MDD Aggregation Module. In this embodiment, the view mechanism of the DBMS enables reference and linking to the data stored in the MDDB of the MDD Aggregation Engine as illustrated in FIG. 6(E). A more detailed description of the view mechanism and the Create View SQL statement may be found in C.J. Date, "An Introduction

to Database Systems," Addison-Wesley, Seventh Edition, 2000, pp. 289-326, herein incorporated by reference in its entirety. Thus, the view mechanism enables the query handler of the DBMS system to forward any SQL query on table TN to the MDD aggregation module via the associated link. In an alternative embodiment, a direct mechanism (e.g., NA trigger mechanism) may be used to enable the DBMS system to reference and link to the data generated by the MDD Aggregation Module and/or stored in the MDDB of the MDD Aggregation Engine as illustrated in FIG. 6F. A more detailed description of trigger mechanisms and methods may be found in C.J. Date, "An Introduction to Database Systems," Addison-Wesley, Seventh Edition, 2000, pp. 250, 266, herein incorporated by reference in its entirety.

In step 609, a user interacts with a client machine to generate a query, and the query is communicated to the query interface. The query interface generate one or more SQL statements. These SQL statements may refer to data stored in tables of the relational datastore, or may refer to the reference defined in step 607 (this reference refers to the data stored in the MDDB of the MDD Aggregation Module). These SQL statement(s) are forwarded to the query handler of the DBMS.

In step 611, the query handler receives the SQL statement(s); and optionally transforms such SQL statement(s) to optimize the SQL statement(s) for more efficient query handling. Such transformations are well known in the art. For example, see Kimball , Aggregation Navigation With (Almost) No MetaData", DBMS Data Warehouse Supplement, August 1996, available at http://www.dbmsmag.com/9608d54.html.

In step 613: the query handler determines whether the received SQL statement(s) [or transformed SQL statement(s)] is on the reference generated in step 607. If so, operation continues to step 615; otherwise normal query handling operations continue in step 625 wherein the relational datastore is accessed to extract, store, and/or manipulate the data stored therein as directed by the query, and results are returned back to the user via the client machine, if needed.

In step 615, the received SQL statement(s) [or transformed SQL statement(s)] is routed to the MDD aggregation engine for processing in step 617 using the link for the reference as described above with respect to step 607.

In step 617, the SQL statement(s) is received by the SQL handler of the MDD Aggregation Module, wherein a set of one or more N-dimensional coordinates are extracted from the SQL statement. In performing this function, SQL handler may utilize an adapter (interface) that maps the data types of the SQL statement issued by query handler of the DBMS (or that maps a standard data type used to represent the SQL statement issued by query handler of the DBMS) into the data types used in the MDD aggregation module.

In step 619, the set of N-dimensional coordinates extracted in step 617 are used by the MDD handler to address the MDDB and retrieve the corresponding data from the MDDB.

Finally, in step 621, the retrieved data is returned to the user via the DBMS (for example, by forwarding the retrieved data to the SQL handler, which returns the retrieved

data to the query handler of the DBMS system, which returns the results of the user-submitted query to the user via the client machine), and the operation ends.

It should be noted that the table data (base data), as it arrives from DBMS, may be analyzed and reordered to optimize hierarchy handling, according to the unique method of the present invention, as described above with reference to Figs. 11A, 11B, and 11C.

Moreover, the MDD control module of the MDD Aggregation Module preferably administers the aggregation process according to the method illustrated in Figs. 9A and 9B. Thus, in accordance with the principles of the present invention, data aggregation within the DBMS can be carried out either as a complete pre-aggregation process, where the base data is fully aggregated before commencing querying, or as a query directed roll-up (QDR) process, where querying is allowed at any stage of aggregation using the "on-the-fly" data aggregation process of the present invention. The QDR process will be described hereinafter in greater detail with reference to Fig. 9C. The response to a request (i.e. a basic component of a client query) requiring "on-the-fly" data aggregation, or requiring access to pre-aggregated result data via the MDD handler is provided by a query/request serving mechanism of the present invention within the MDD control module, the primary operations of which are illustrated in the flow chart of Fig. 6D. The function of the MDD Handler is to handle multidimensional data in the storage(s) module in a very efficient way, according to the novel method of the present invention, which will be described in detail hereinafter with reference to Figs. 10A and 10B.

The SQL handling mechanism shown in Fig. 6D is controlled by the MDD control module. Requests are queued and served one by one. If the required data is already pre-calculated, then it is retrieved by the MDD handler and returned to the client. Otherwise, the required data is calculated "on-the-fly" by the aggregation engine, and the result moved out to the client, while simultaneously stored by the MDD handler, shown in Fig. 6C.

As illustrated in Fig. 19G, the DBMS of the present invention as described above may be logically partitioned into a relational part and a non-relational part. The relational part includes the relational datastore (e.g., table(s) and dictionary) and support mechanisms (e.g., query handling services). The non-relational part includes the MDD Aggregation Module. As described above, bi-directional data flow occurs between the relational part and the non-relational part as shown. More specifically, during data load operations, data is loaded from the relational part (i.e., the relational datastore) into the non-relational part, wherein it is aggregated and stored in the MDDB. And during query servicing operations, when a given query references data stored in the MDDB, data pertaining to the query is generated by the non-relational part (e.g., generated and/or retrieved from the MDDB) and supplied to the relational part (e.g., query servicing mechanism) for communication back to the user. Such bi-directional data flow represents an important distinguishing feature with respect to the prior art. For example, in the prior art MOLAP architecture as illustrated in Fig. 1B, unidirectional data flows occurs from the relational data base (e.g., the Data Warehouse RDBMS system) into the MDDB during data loading operations.

Figs. 20A and 20B outline two different implementations of the DBMS of the present invention. In both implementations, the query handler of the DBMS system supplies aggregated results retrieved from the MDD to a client.

Fig. 20A shows a separate-platform implementation of the DBMS system of the illustrative embodiment shown in Fig. 19A, wherein the relational part of the DBMS reside on a separate hardware platform and/or OS system from that used to run the non-relational part (MDD Aggregation Module). In this type of implementation, it is even possible to run parts of the DBMS system and the MDD Aggregation Module on different-type operating systems (e.g. NT, Unix, MAC OS).

Fig. 20B shows a common-platform implementation of the DBMS system of the illustrative embodiment shown in Fig. 20A, wherein the relational part of the DBMS share the same hardware platform and operating system (OS) that is used to run the non-relational part (MDD Aggregation Module).

Fig. 21 shows the improved DBMS (e.g., RDBMS) of the present invention as a component of a data warehouse, serving the data storage and aggregation needs of a ROLAP system (or other OLAP systems alike). Importantly, the improved DBMS of the present invention provides flexible, high-performance access and analysis of large volumes of complex and interrelated data. Moreover, the improved Data Warehouse DBMS of the present invention can simultaneously serve many different kinds of clients (e.g. data mart, OLAP, URL) and has the power of delivering an enterprise-wide data storage and aggregation in a cost-effective way. This kind of system eliminates redundancy over the group of clients, delivering scalability and flexibility. Moreover, the improved DBMS of the present invention can be used as the data store component of in any informational database system as described above, including data analysis programs such as spread-sheet modeling programs, serving the data storage and aggregation needs of such systems.

Fig. 22 shows an embodiment of the present invention wherein the DBMS (e.g., RDBMS) of the present invention is a component of a data warehouse - OLAP system. The DBMS operates as a traditional data warehouse, serving the data storage and aggregation needs of an enterprise. In addition, the DBMS includes integrated OLAP Analysis Logic (and preferably an integrated Presentation Module not shown) that operates cooperatively with the query handling of the DBMS system and the MDD Aggregation Module to enable users of the DBMS system to execute multidimensional reports (e.g., ratios, ranks, transforms, dynamic consolidation, complex filtering, forecasts, query governing, scheduling, flow control, pre-aggregate inferencing, denomalization support, and/or table partitioning and joins) and preferably perform traditional OLAP analyses (grids, graphs, maps, alerts, drill-down, data pivot, data surf, slice and dice, print). Importantly, the improved DBMS of the present invention provides flexible, high-performance access and analysis of large volumes of complex and interrelated data. Moreover, the improved DBMS of the present invention can simultaneously serve many different kinds of clients (e.g. data mart, other OLAP systems, URL-Directory Systems) and has the power of delivering enterprise-wide data storage and aggregation and OLAP analysis in a cost-effective way. This kind of system eliminates

redundancy over the group of clients, delivering scalability and flexibility. Moreover, the improved DBMS of the present invention can be used as the data store component of in any informational database system as described above, serving the data storage and aggregation needs of such systems.

Functional Advantages Gained By The Improved DBMS Of The Present Invention

The features of the DBMS of the present invention, provides for dramatically improved response time in handling queries issued to the DBMS that involve aggregation, thus enabling enterprise-wide centralized aggregation. Moreover, in the preferred embodiment of the present invention, users can query the aggregated data in an manner no different than traditional queries on the DBMS.

The method of Segmented Aggregation employed by the novel DBMS of the present invention provides flexibility, scalability, the capability of Query Directed Aggregation, and speed improvement.

Moreover, the method of Query Directed Aggregation (QDR) employed by the novel DBMS of the present invention minimizes the data handling operations in multi-hierarchy data structures, eliminates the need to wait for full aggregation to be complete, and provides for build-up of aggregated data required for full aggregation.

It is understood that the System and Method of the illustrative embodiments described herein above may be modified in a variety of ways which will become readily apparent to those skilled in the art of having the benefit of the novel teachings disclosed herein. All such modifications and variations of the illustrative embodiments thereof shall be deemed to be within the scope and spirit of the present invention as defined by the Claims to Invention appended hereto.